Imperial College London

The Next 700 **Domain Specific Languages**

Nicolas Wu

Imperial College London

Peter Landin Seminar, BCS 2022







functional





Alan Turing

Before Computers







"When trying to get from England to Holland, I was detained for some time peral C. 3. ' in the Thames by adverse winds. During that time, knowing not what to do, especially thought about my old design of a rational language [...]" and baving nobody in the ship except sailors, I meditated on things, and Bui "For if we bad such as I imagine, we could reason about metaphysics 1890 and morality just like we do about geometry and analysis [...]" Beibmanniche

letter from Leibniz to Jean Galois, 1677

Combinatio

DISSERTATIO Leibniz, 1666 ARTE COMBI-NATORIA, Ex Arithmeticz fundamentis Complicationum ac Transpositionum Doctrina novis praceptis exftruitur, & ufus ambarum per universum scientiarum orbem ostenditur; nova etiam Artis Meditandi , Logicæ Inventionis femina sparguntur. Prafixa est Synopfis totius Tractatus, & additamenti loco ETENTIA DEL Leibniz, The Art of Combinations(1666) GOTT This extended doctoral thesis was about: LH • a proof of the existence of God from some APUI hypotheses and axioms about motion · an alphabet of human thought, the "alphabetum cogitationum humanarium" the idea that concepts are combinations of a core set of concepts

"For if praise is given to the men who have determined the number of regular solids [...] how much better will it be to bring under mathematical laws human reasoning, which is the most excellent and useful thing we have." (Davis, 2018)







Kurt Hödel 1906 - 1978

Gödel (1931) showed that any reasonable system of logic has true statements that cannot be proven within that system

> but if the Entscheidungsproblem holds true, then perhaps an algorithm can prove statements?

1931

Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I¹).

Von Kurt Gödel in Wion.

Die Entwicklung der Mathematik in der Richtung zu größerer Exaktheit hat bekanntlich dazu geftihrt, daß weite Gebiete von ihr formalisiert wurden, in der Art, daß das Beweisen nach einigen wenigen mechanischen Regeln vollzogen werden kann. Die umfassendsten derzeit aufgestellten formalen Systeme sind das System der Principia Mathematica (PM)²) einerseits, das Zermelo-Fraenkelsche (von J. v. Neumann weiter ausgebildete) Axiomensystem der Mengenlehre³) andererseits. Diese beiden Systeme sind so weit, daß alle heute in der Mathematik angewendeten Beweismethoden in ihnen formalisiert, d. h. auf einige wenige Axiome und Schlußregeln zuräckgeführt sind. Es liegt daher die Vermutung nahe, daß diese Axiome und Schlußregeln dazu ansreichen, alle mathematischen Fragen, die sich in den betreffenden Systemen tiberhaupt formal ausdrücken lassen, auch zu entscheiden. Im folgenden wird gezeigt, daß dies nicht der Fall ist, sondern daß es in den beiden angeführten Systemen sogar relativ einfache Probleme aus der Theorie der gewöhnlichen ganzen Zahlen gibt4), die sich aus den Axiomen nicht

') Vgl. die im Anzeiger der Akad. d. Wiss. in Wien (math.-naturw. Kl.) 1930, Nr. 19 erschienene Zusammenfassung der Resultate diesor Arbeit.

2) A. Whitehead und B. Russell, Principia Mathematica, 2. Aufl., Cambridge 1925. Zu den Axiomen des Systems PM rechnen wir insbesondere

THE JOURNAL OF SYMBOLIC LODIC Volume 1, Number 1, March 1936



In a recent paper¹ the author has proposed a definition of the commonly used term "effectively calculable" and has shown on the basis of this definition that the general case of the Entscheidungsproblem is unsolvable in any system of symbolic logic which is adequate to a certain portion of arithmetic and is ω -consistent. The purpose of the present note is to outline an extension of this result to the engere Funktionenkalkül of Hilbert and Ackermann.²

In the author's cited paper it is pointed out that there can be associated recursively with every well-formed formula² a recursive enumeration of the formulas into which it is convertible.⁹ This means the existence of a recursively defined function a of two positive integers such that, if y is the Gödel representation of a well-formed formula Y then a(x, y) is the Gödel representation of the xth formula in the enumeration of the formulas into which Y is convertible.

Consider the system L of symbolic logic which arises from the engere Funktionenkalkül by adding to it: as additional undefined symbols, a symbol 1 for the number 1 (regarded as an individual), a symbol = for the propositional function = (equality of individuals), a symbol s for the arithmetic function x+1, a symbol a for the arithmetic function a described in the preceding paragraph, and symbols b_1, b_2, \cdots, b_k for the auxiliary arithmetic functions which are employed in the recursive definition of a and as addition

alonzo Church

Church (1936) and Turing (1936)

Independently answered the

Entscheidungsproblem:

No! Such an algorithm is not possible

A. M. TURING

alan Turing

[Nov. 12,

A NOTE ON THE ENTSCHEIDUNGSPROBLEM

ALONZO CHURCH

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TUBING.

[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers. it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable



The Lambda Calculus

alonzo Church

λ-calculus is about bindings and substitutions Instead of

HILL SHE SHE

$f = \lambda x \cdot 5x + 3$ A SET OF POSTULATES FOR THE FOUNDATION OF LOGIC.

f(x) = 5x + 3

We can isolate f:

By ALONZO CHURCH.2

1932

oduction. In this paper we present a set of postulates for the of formal logic, in which we avoid use of the free, or real, and in which we introduce a certain restriction on the law of med middle as a means of avoiding the paradoxes connected with the mathematics of the transfinite.

Our reason for avoiding use of the free variable is that we require that

every combination of symbols belonging to our system, if it represents a proposition at all, shall represent a particular proposition, unambigouously, and without the addition of verbal explanations. That the use of the free variable involves violation of this requirement, we believe is readily seen. For example, the identity

(1)

$$a(b+c) = ab + ac$$

in which a, b, and c are used as free variables, does note state a definite proposition unless it is known what values may be taken on by these variables, and this information, if not implied in the context, must be a

AN UNSOLVABLE PROBLEM OF ELEMENTARY NUMBER THEORY.¹

The definition of "effectively calculable"

Church numerals:

 $1 = \lambda SZ.SZ$ $2 = \lambda SZ.S(SZ)$ $3 = \lambda SZ.S(S(SZ))$

 $m + n = \lambda SZ.mS(nSZ)$ $_+$ = λ m n s z . m s (n s z)

1936

By ALONZO CHURCH.

1. Introduction. There is a class of problems of elementary number theory which can be stated in the form that it is required to find an effectively calculable function f of n positive integers, such that $f(x_1, x_2, \cdots, x_n) = 2^{\frac{n}{2}}$ is a necessary and sufficient condition for the truth of a certain proposition of elementary number theory involving x_1, x_2, \cdots, x_n as free variables.

An example of such a problem is the problem to find a means of determining of any given positive integer n whether or not there exist positive integers x, y, s, such that $x^n + y^n = z^n$. For this may be interpreted, required to find an effectively calculable function f, such that f(n) is equal to 2 if and only if there exist positive integers x, y, z, such that $x^n + y^n = z^n$. Clearly the condition that the function f be effectively calculable is an essential part of the problem since without it the problem becomes trivial

A proof that converting a term A into B is undecidable

THE JOURNAL OF SYNCBOLIC LOGIC Volume 1, Number 1, March 1935

A NOTE ON THE ENTSCHEIDUNGSPROBLEM ALONZO CHURCH

In a recent paper¹ the author has proposed a definition of th used term "effectively calculable" and has shown on the basis of t that the general case of the Entscheidungsproblem is unsolvable in of symbolic logic which is adequate to a certain portion of arith ω -consistent. The purpose of the present note is to outline an ext result to the engere Funktionenkalkül of Hilbert and Ackermann. In the author's cited paper it is pointed out that there can

recursively with every well-formed formula³ a recursive enumera mulas into which it is convertible.³ This means the existence of defined function a of two positive integers such that, if y is the tation of a well-formed formula Y then a(x, y) is the Gödel repre xth formula in the enumeration of the formulas into which Y is Consider the system L of symbolic logic which arises from t

tionenkalkül by adding to it: as additional undefined symbols the number 1 (regarded as an individual), a symbol = for function = (equality of individuals), a symbol s for the ar x+1, a symbol a for the arithmetic function a described in th graph, and symbols b_1, b_2, \cdots, b_k for the auxiliary arithmetic comployed in the recursive definition of a; and as additional







Turing Machines

Automatic machines.

If at each stage the motion of a machine (in the sense of §1) is completely determined by the configuration, we shall call the machine an "automatic machine" (or a-machine).

Computing machines.

If an a-machine prints two kinds of symbols, of which the first kind (called figures) consists entirely of 0 and 1 (the others being called symbols of the second kind), then the machine will be called a computing machine.

230

A. M. TURING

[Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

[Received 28 May, 1936.-Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include

We can show further that there can be no machine & which, when supplied with the S.D of an arbitrary machine \mathcal{M} , will determine whether \mathcal{M} ever prints a given symbol (0 say).

The results of §8 have some important applications. In particular, they can be used to show that the Hilbert Entscheidungsproblem can have no solution. For the present I shall confine myself to proving this particular

Turing showed construction of a formula that takes in a Turing machine as input and is provable only if the machine halts

but he also showed that no machine can determine if an arbitrary machine halts





Contract No. W-670-ORD-4920 Between the

United States Army Ordnance Department

Samuel I alexander

first Drift of a Report

John von Neumann

and the

University of Fennsylvania

Moore School of Electrical Engineering University of Pennsylvania

June 30, 1945

National Bureau of Standards Division 12 Data Processing Systems

1945

2.1



Early Programming Languages



FORTRAN, 1954

C - FOR CONNENT	VINUATION	FORTRAN STATEMENT		
NUMBER	Con			
		REWIND 3		
		DO 3 I = 1,N		
	_	DO 3 J = 1, N		
		IF(A(1,J)-A(J,1)) 3,20,3		
3		CONTINUE		
		END FILE 3		
	_	MORE PROGRAM		
20		IF(I-J) 21,3,21		
21		WRITE TAPE 3, I, J, A(I, J)		
i l	_	GO TO 3		



John Backup

Fortran



- - -

INSTRUCTION COUNTER

7.7

John Backus in "Think, IBM, July/August 1979"





LISP, 1958



John McCarthy

LISP I PROGRAMMER'S MANUAL March 1, 1960

Artificial Intelligence Group

J. McCarthy

R. Brayton

D. Edwards

P. Fox

L. Hodes

D. Luckham

K. Maling

D. Park

S. Russell

If ξ is a form in variables x_1, \ldots, x_n , then $\lambda((x_1, \ldots, x_n), \xi)$ will be taken to be the function of n variables whose value is determined by substituting the arguments for the variables x_1, \ldots, x_n in that order in \mathcal{E} and evaluating the resulting expression. For example, $\lambda((x,y),y^2+x)$ is a function of two variables, and $\lambda((x,y),y^2+x)(3,4) = 19$.

> Conditional expressions are a device for expressing the dependence of quantities on propositional quantities. A conditional expression has the form

 $(p_1 \rightarrow e_1, \dots, p_n \rightarrow e_n)$ where the p's are propositional expressions and the e's are expressions of any kind. It may be read, "If p1 then e1, otherwise if p2 then e2,..., otherwise if pn then en," or "p1 yields e1,...,p yields en."

By using conditional expressions we can, without circularity, define functions by formulas in which the defined function occurs. For example, we write

 $n! = (n=0 \rightarrow 1, T \rightarrow n \cdot (n-1)!)$

When we use this formula to evaluate O! we get the answer 1;

g. Functions with Functions as Arguments

There are a number of useful functions some of whose arguand the second ments are functions. They are especially useful in defining other functions. One such function is maplist[x;f] with an Sexpression argument x and an argument f that is a function from S-expressions to S-expressions. We define $maplist[x;f] = [null[x] \rightarrow NIL;T \rightarrow cons[f[x];maplist[cdr[x];f]]]$

Features of Lisp

- Based on lambda calculus
- Conditionals
- Recursive functions
- List processing (car, cdr, cons)
- Higher-order functions

An S-expression x that is not atomic is represented by a word, the address and decrement parts of which contain the locations of the subexpressions car[x] and cdr[x], respectively. In the list form of expressions the S-expression (A, (B, C), D) is represented by the list structure of Fig. 2.



When a list structure is regarded as representing a list, we see that each term of the list occupies the address part of a word, the decrement part of which points to the word containing the next term, while the last word has NIL in its decrement. The dot notation, e.g. (A+B), which is discussed in Section 2.2 is not allowed in LISP I; all lists and sublists must end with .





Peter naur 1928 - 2016



John Backus 1924 - 2007



John McCarthy

... and Green, Katz, Perlis, Rutishauser, Samelson, van Mijngaarden, Vauguois, Wegstein, and Woodger

REVISED REPORT ON THE ALGORITHMIC LANGUAGE ALGOL 60

Dedicated to the memory of William Turanski

J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy P. Naur, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauguois, J. H. Wegstein.

> Edited by Peter Naur

Approved by the council of the International Federation

procedure Absmax(a)size:(n,m)Result:(y)Subscripts: (i,k) comment The absolute greatest element of the matrix a, of size n by m is transferred to y, and the subscripts of this element to i and k ; arraya ; integer n, m, i, k ; real y ; begin integer p, q ; y := 0; for p := 1 step 1 until n do for q := 1 step 1 until m do if abs(a[p,q]) > y then begin y:=abs(a[p,q]); i:=p; k := qend end Absmax

Algol 1958

Algol

- · Iteration and recursion*
- Distinct assignment (:=) and equality (=)
- · Code blocks and lexical scope
- · Nested procedure, and procedure passing

·BNF

· Call-by-value and call-by-name

"... a language so far ahead of its time, that it was not only an improvement on its predecessors, but also on nearly all its successors" - Tony Hoare, 1974, in Hints on Programming Language Design



Here is a typical form of AE, presented both informally and formally.

let a = Aand b = Band f(x, y) = Flet rec g(x) = Gand h(y, z) = Hlet k(u, v)(x) = K

The only consideration in choosing between let and where will be the relative convenience of writing an auxiliary definition before or after the expression it qualifies.

ALGOL 60

Identifiers, operator symbols, also some special words and configurations Local identifiers

Formal parameters

designator, Function scripted variable, and procedure statement Procedure Actual parameter called by

name

SECD Machine First abstract machine for evaluating lambda expressions

The mechanical evaluation of expressions

By P. J. Landin

1964

This paper is a contribution to the "theory" of the activity of using computers. It s some forms of expression used in current programming languages can be modelled in λ-notation, and then describes a way of "interpreting" such expressions. This sa method, of analyzing the things computer users write, that applies to many different orientations and to different phases of the activity of using a computer. Also a tec introduced by which the various composite information structures involved can be characterized in their essentials, without commitment to specific written or other represe

Introduction

The point of departure of this paper is the idea of a machine for evaluating schoolroom sums, such as

1.
$$(3 + 4)(5 + 6)(7 + 8)$$

2. If
$$2^{15} < 3^{12}$$
 then $1^2\sqrt{2}$ else $5^3\sqrt{2}$

3. $\sqrt{\left(\frac{17\cos\pi/17 - \sqrt{(1-17\sin\pi/17)}}{17\cos\pi/17 + \sqrt{(1+17\sin\pi/17)}}\right)}$

Any experienced computer user knows that his activity scarcely resembles giving a machine a numerical expression and waiting for the answer. He is involved with flow diagrams, with replacement and sequencing, with programs, data and jobs, and with input and output. There are good reasons why current information-

is written explicitly and prefixed to its operand(each operand (or operand-list) is enclosed in br e.g.

 $/(a, + (\times (2, b), 3)).$

This notation is a sort of standard notation in a all the expressions in this paper could (with some of legibility) be rendered.

The following remarks about applicative structure will be illustrated by examples in which an expression is written in two ways: on the left in some notation whose applicative structure is being discussed, and on the right in a form that displays the applicative structure more

Algol and Lambdas, 1965

 $\{\lambda(a, b, f).$ $\{\lambda(g,h).$ $\{\lambda k.X\}[\lambda(u, v).\lambda x.K]\}$ $[Y\lambda(g, h).(\lambda x.G, \lambda(y, z).H]]$ $[A, B, \lambda(x, y).F]$

TABLE 1

IAEs

Identifiers

Variables bound in a λ -expression occurring as operator Variables bound in a λ -expression occurring as operand Operator/operand combination sub-

> λ -expression λ ()-expression

Applicative expressions (AE) = Syntactic sugar for lambda expressions Imperative AE (IAE) = AE + assignment operation (IAE)



A Correspondence Between ALGOL 60 and Church's Lambda-Notation: Part I*

----- Standards

Br P. J. LANDINT

This paper describes how some of the semantics of ALGOL 60 can be formalized by establishing a correspondence between expressions of ALGOL 60 and expressions in a modified form of Church's λ -notation. First a model for camputer languages and computer behavior is described, based on the notions of functional application and functional abstraction, but also having analogues for imperative language features. Then this model is used as an "abstract object language" into which ALGOL 60 is mapped. Many of ALGOL 60's features emerge as particular arrangements of a small number of structural rules, suggesting new classifications and generalizations. The correspondence is first described informally, mainly by

illustrations. The second port of the paper gives a format description, i.e. an "abstract compiler" into the "abstract object language." This is itself presented in a "purely functional" notation, that is one using only application and abstraction.

Contents

The Constants and Primitices of

Introduction

Anyone familiar with both Church's x-calculi [7]) and ALCOL 60 [5] will have noticed a super semblance between the way variables tie up wit in a nest of λ -expressions, and the way identified with the headings in a nest of procedures and bloc may also have observed that in, say $\{\lambda f f(a) + f(b)\}[\lambda x.x^2 + px + q]$

the two λ -expressions, i.e. the operator and the play roughly the roles of block-body and declaration, respectively. The present paper exresemblance in some detail. The presentation falls into four sections. Th

tion, following this introduction, gives some for examining the correspondence. The second scribes an abstract language based on Church This abstract language is a development of the system presented in [3] and some acquaintance paper (hereinafter referred to as [MEE]) is as The third section describes informally, main trations, a correspondence between expressio 60 and expressions of the abstract language section formalizes this correspondence; it fi a sort of "abstract ALGOL 60" and then prese tions that map expressions of abstract Algo the one hand, Algor 60 texts, and on the expressions of the abstract language.

Motivation It seems possible that the correspondence the basis of a formal description of the sema-60.1 As presented here it reduces the problem amonties to that of specifying th



LOHSE, Asst. Editor, Information Interch. V. SMITH, Asst. Editor. Programming Lat



Peter Landin 1930 - 2009

aka Church without Lambda

The Next 700 Programming Languages

P. J. Landin

1966

Univac Division of Sperry Rand Corp., New York, New York

"... today ... 1,700 special programming languages used to 'communicate' in over 700 application areas."-Computer Software Issues, an American Mathematical Association Prospectus, July 1965.

A family of unimplemented computing languages is described that is intended to span differences of application area by a unified framework. This framework dictates the rules about the uses of user-coined names, and the conventions about characterizing functional relationships. Within this framework the design of a specific language splits into two independent parts. One is the choice of written appearances of programs (or more generally, their physical representation). The other is the choice of the abstract entities (such as numbers, character-strings, lists of them, functional relations among them) that can be referred to in the language.

The system is biased towards "expressions" rather than "statements." It includes a nonprocedural (purely functional) subsystem that aims to expand the class of users' needs that can be met by a single print-instruction, without sacrificing the important properties that make conventional right-hand-side expressions easy to construct and understand.

1. Introduction

Most programming languages are partly a way of expressing things in terms of other things and partly a basic set of given things. The Iswiw (If you See What I Mcan) system is a byproduct of an attempt to disentangle these two aspects in some current languages.

This attempt has led the author to think that many linguistic idiosyncracies are concerned with the former

rather than the latter, whereas aptitude for a particular class of tasks is essentially determined by the latter rather than the former. The conclusion follows that many language characteristics are irrelevant to the alleged

Iswim is an attempt at a general purpose system for describing things in terms of other things, that can be problem-oriented by appropriate choice of "primitives," So it is not a language so much as a family of languages, of which each member is the result of choosing a set of primitives. The possibilities concerning this set and what

is needed to specify such a set are discussed below. Iswim is not alone in being a family, even after syntactic variations have h

differences in the set of things provided by the library or operating system. Perhaps had ALGOL 60 been launched as a family instead of proclaimed as a language, it would have fielded some of the less relevant criticisms of its

At first sight the facilities provided in Iswim will appear comparatively meager. This appearance will be especially misleading to someone who has not appreciated how much of current manuals are devoted to the explanation of common (i.e., problem-orientation independent) logical structure rather than problem-oriented specialties. For example, in almost every language a user can coin names, obeying certain rules about the contexts in which the name is used and their relation to the textual segments that introduce, define, declare, or otherwise constrain its use. These rules vary considerably from one language to another, and frequently even within a single language there may be different conventions for different classes of names, with near-analogies that come irritatingly close to being exact (Note that restrictions on what names can be coined also vary, but these are trivial differences. When they have any logical significance it is likely to be perni-

cious, by leading to puns such as ALGOL's integer labels.) So rules about user-coined names is an area in which we might expect to see the history of computer applications give ground to their logic. Another such area is in specifying functional relations. In fact these two areas are closely related since any use of a user-coined name implicitly involves a functional relation; e.g., compare

where x = b + 2cj(b+2e)

Iswim is thus part programming language and part program for research. A possible first step in the research program is 1700 doctoral theses called "A Correspondence between x and Church's λ -notation."

2. The where-Notation

In ordinary mathematical com-



Church without Lambda) [7 - 3]where notation •More convenient that lambda terms Indentation sensitive •Multiple and recursive definitions Used for types The Next 700 Programming Univac Division of Sperry Rand Corp., New Y 1)]. The mechanical evaluation of expressions By P. J. Landin vays I am in orientations and to different phases of the activity introduced by which the various composite information characterized in their essentials, without commitment to specific written or other r would regret ingement of Introduction The point of departure of this paper is the idea of a e.g. machine for evaluating schoolroom sums, such as Peter Naur 1928 - 2016 1. (3 + 4)(5 + 6)(7 + 8)ting 2. if $2^{19} < 3^{12}$ then ${}^{12}\sqrt{2}$ else ${}^{53}\sqrt{2}$ 3. $\sqrt{\left(\frac{17\cos\pi/17-\sqrt{(1-17\sin\pi/17)}}{17\cos\pi/17+\sqrt{(1+17\sin\pi/17)}}\right)}$

$$(u-1)(u+2) \quad \{\lambda u \cdot (u-1)(u+2)\}$$
where $u = 7 - 3$.

$$u(u+1) - v(v+1) \quad \{\lambda(u,v) \cdot u(u+1) - v(v+1)\}$$
where $u = 2p + q$ $[2p + q, p - 2q]$
and $v = p - 2q$.

$$f(7-3) \quad \{\lambda f. f(7-3)\}$$
where rec $f(n) =$ $[Y\lambda f. \lambda n. \text{ if } n = 0 \text{ then } 1$
else $nf(n-1)$

$$where n = round(n)$$
instead of the specification

$$Regarding indentation, in many ussympathy with this, but [..] you usit because of the kind of rearrantmanuscripts done in print$$

Any experienced computer user knows that his

of legibility) be rendered. The following remarks about applic will be illustrated by examples in which a written in two ways: on the left in some





is written explicitly and prefixed to its op each operand (or operand-list) is enclosed

 $((a, + (\times (2, b), 3))).$

This notation is a sort of standard nota all the expressions in this paper could (v

The "physical/logical" terminology is often used to distinguish features that are a fortuitous consequence of physical conditions from features that are in some sense more essential. This idea is carried further by making a similar distinction among the "more essential" features. In fact ISWIM is presented here as a four-level concept comprising the following: tokenize

he Next 700 Programming Languages

P. J. Landin

Univac Division of Sperry Rand Corp., New York, New York

"... today ... 1,700 special programming languages used to 'communicate' in over 700 application areas."-Computer Software Issues, an American Mathematical Association Prospectus, July 1965.

ented computing languages is despan differences of application area This framework dictates the rules cined names, and the conventions onal relationships. Within this framecific language splits into two indechoice of written appearances of ally, their physical representation).

1964

e abstract er them, functi to in the lo wards "ex nonproced and the d t-instructio cke con and unde

SUSTAN

differences in the set of things provided by the library or operating system. Perhaps had ALGOL 60 been launched as a family instead of proclaimed as a language, it would have fielded some of the less relevant criticisms of its

At first sight the facilities provided in Iswim will appear comparatively meager. This appearance will be especially misleading to someone who has not appreciated how much of current manuals are devoted to the explanation of ation independent) logical

The mechanical evaluation of expressions

This paper is a contribution to the "theory" of the activity of using computers. It shows how some forms of expression need in current programming languages can be modelled in Church's Interpreter to a contribution to the "theory" of the activity of using computers. If shows how some forms of expression used in current programming languages can be modelled in Church's A-notation, and then describes a way of "interpreting" such expressions. This suggests a some forms of expression used in current programming languages can be modelled in Church's λ -notation, and then describes a way of "interpreting" such expressions. This suggests a method, of analyzing the things computer users write, that applies to many different problem A-notation, and then describes a way of "interpreting" such expressions. This suggests a method, of analyzing the things computer users write, that applies to many different problem orientations and to different phases of the activity of using a computer. Also a technique is method, of analyzing the things computer users write, that applies to many different problem orientations and to different phases of the activity of using a computer. Also a technique is introduced by which the various composite information structures involved can be formally orientations and to different phases of the activity of using a computer. Also a technique is introduced by which the various composite information structures involved can be formally characterized in their essentials, without commitment to specific written or other representations.

The point of departure of this paper is the idea of a machine for evaluating schoolroom sums, such as 2. if 219 < 312 then 12./2 also

parse

interpret

evaluate

is written explicitly and prefixed to its operand/ each operand (or operand-lier) is

Language Abstraction f(b + 2c) Physical ISWIM where f(x) = x(x+a)f(b+2c)where f(x) = x(x+a)Logical ISWIM Abstract ISWIM Applicative Expression 1/10th of the paper is devoted to the idea of abstraction layers for defining a language! Value Today this is regarded as the standard means of giving language semantics







Enabling equational reasoning featured heavily in this paper

(1) the extent to which a subexpression can be replaced by an equivalent subexpression without disturbing the equivalence class of the whole expression. Without this group the other rules would be applicable only to complete

expressions, not to subexpressions. (2) user-coined names, i.e., in definitions and, in particu-

(3) built-in entities implicit in special forms of exlar, function definitions.

pression. The only instances of this in Iswim are conditional expressions, listings and self-referential definitions. (4) named entities added in any specific problem-

orientation of Iswim.

(let)

Iswim is an attempt at a general purpose system for describing things in terms of other things, that can be problem-oriented by appropriate choice of "primitives." So it is not a language so much as a family of languages, of which each member is the result of choosing a set of primitives. The possibilities concerning this set and what

ה. המדררר	00000		
Jurvar	rences		
(1): subexpressions	S		
(μ) Ι	$\overline{\mathbf{f}} L \equiv L'$	then L ($M) \equiv L'(M)$ $M = L(M')$
(v) _	$M \equiv M$	unen D	
(2): definitions			
(let) le	$\mathbf{t} \ x = M; \ L$	<u> </u>	L where $x =$
(3): built-in	expressions		
(→) true →') false 	M; N M: N	$ = M \\ \equiv N $
	/		

(4): primitives

A problem-orientation of ISWIM can be characterized by additional axioms. In the simplest case such an axiom is an ISWIM definition. The resulting modification is called a "definitional extension" of the original system.



The commonplace expressions of arithmetic and algebra have a certain simplicity that most communications to computers lack. In particular, (a) each expression has a nesting subexpression structure, (b) each subexpression denotes something (usually a number, truth value or numerical function), (c) the thing an expression denotes, i.e., its "value", depends only on the values of its subexpressions, not on other properties of them. It is these properties, and crucially (c), that explains

why such expressions are easier to construct and understand. Thus it is (c) that lies behind the evolutionary trend towards "bigger righthand sides" in place of strings of small, explicitly sequenced assignments and jumps.

> The importance of denotational semantics was clearly highlighted



Iswim brings into sharp relief some of the distinctions that the author thinks are intended by such adjectives as procedural, nonprocedural, algorithmic, heuristic, imperative, declarative, functional, descriptive. Here is a suggested classification, and one new word.

denotative!

Peter Landin

functional















The Next 700 <u>Frogranning</u> Languages

Datastructures and Recursion Schemes

Lists can be evaluated by folding their structure: cons and nil are replaced with a semantics of what to do

> This technique generalises to many tree-like datatypes

There are many variations on this theme, depending on mutual recursion, recursion with parameters, recursion with historic context, and many more

$\begin{bmatrix} 2 & 7 & 1 & 8 \end{bmatrix}$ foldr (+) 0 (2 : 7 : 1 : 8 : []) = 2 + 7 + 1 + 8 + 0 = 18





catamorphism

 Δ





mutumorphism

and the second second





zygomorphism 24

anamorphism



paramorphism

No. of Concession, Name

histomorphism

PL OF

futumorphism

Free^M – U^M

dynamorphism



Datastructures and Recursion Schemes

Lists can be evaluated by folding their structure: cons and nil are replaced with a semantics of what to do

> This technique generalises to many tree-like datatypes

There are many variations on this theme, depending on mutual recursion, recursion with parameters, recursion with historic context, and many more

2013

Unifying Structured Recursion Schemes

Ralf Hinze Nicolas Wu Jeremy Gibbons

Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, England

Abstract

Folds over inductive datatypes are well understood and widely used. In their plain form, they are quite restricted; but many disparate generalisations have been proposed that enjoy similar calenlational benefits. There have also been attempts to unify the various generalisations: two prominent such unifications are the 'recursion schemes from comonads' of Uustalu, Vene and Pardo, and our own 'adjoint folds'. Until now, these two unified schemes have appeared incompatible. We show that this appearance is illusory: in fact, adjoint folds subsume recursion schemes from comonads. The proof of this claim involves standard constructions in category theory that are nevertheless not well known in functional pro-

paramorphisms [21], in which the body of structural recurs access to immediate subterms as well as to their images ur recursion; histomorphisms [26], in which the body has acces recursive images of all subterms, not just the immediate or so-called generalised folds [4], which use polymorphic rec to handle nested datatypes.

The many divergent generalisations of catamorphisms e bewildering to the uninitiated, and there have been attempts to them. One approach is the identification of recursion schemes comonads [30] (which we call 'rsfes' for short). Comon

Or: The Mother of All Structured Recursion Schemes

Ralf Hinze Nicolas Wu Jeremy Gibbons Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford, OX1 3QD, England {ralf.hinze,nicolas.wu,jereny.gibbons}@cs.ox.ac.uk

Abstract

The past decades have witnessed an extensive study of structured recursion schemes. A general scheme is the hylomorphism, which captures the essence of divide-and-conquer: a problem is broken into sub-problems by a coalgebra; sub-problems are solved recursively; the sub-solutions are combined by an algebra to form a solution. In this paper we develop a simple toolbox for assembling recursive coalgebras, which by definition ensure that their hylo equations have unique solutions, whatever the algebra. Our main tool is the conjugate rule, a generic rule parametrized by an adjunction and a conjugate pair of natural transformations. We show that many basic adjunctions induce useful recursion schemes. In fact, almost every structured recursion scheme seems to arise as an instance

[2, 7, 1, 8] foldr (+) 0 (2 : 7 : 1 : 8 : []) = 2 + 7 + 1 + 8 + 0 = 18



Conjugate Hylomorphisms

even in Haskell, accidentally writing a non-terminating dynamic programming algorithm is all too easy using explicit recursion, when this can be prevented by using an appropriate recursion scheme. Canonical examples are provided by folds (catamorphisms), which consume data structures, and unfolds (anamorphisms), which produce them, solutions h and k of the following recursion equations:

> $h \cdot in = a \cdot Fh$ $out \cdot k = F k \cdot c$,

which are shaped by a base functor F. Categorically, these recursion schemes arise from constructions in which in and out are initial algebras and final coalgebras respectively. Initiality and finality each correspond to both existence and uniqueness of solutions to the equations above, traditionally written $h = \langle a \rangle$ and $k = \langle c \rangle$

catamorphism

2022

Fantastic Morphisms and Where to Find Them A Guide to Recursion Schemes

Zhixuan Yang and Nicolas Wu Imperial College London, United Kingdom (a.yang20,n.wu)@imperial.ac.uk

Abstract. Structured recursion schemes have been widely used in constructing, optimizing, and reasoning about programs over inductive and coinductive datatypes. Their plain forms, catamorphisms and anamorphiams, are restricted in expressiveness. Thus many generalisations have passaw, are restricted in expressiveness. This many selections mave been proposed, which further lead to several unifying frameworks of been proposed, which further lead to several unitying traneworks of structured recursion schemes. However, the existing work on unifying frameworks typically focuses on the categorical foundation, and thus is perhaps inaccessible to practitioners who are willing to apply recursion as introduces structured recursion schemes from



Domain-Specific Languages

syntax tree + recursion scheme

2014

Folding Domain-Specific Languages: Deep and Shallow Embeddings

(Functional Pearl)

Jeremy Gibbons Nicolas Wu Department of Computer Science, University of Oxford (joramy.gibbons,nicolas.wu)@as.ox.ac.uk

Abstract

A domain-specific language can be implemented by embedding within a general purpose hest language. This embedding may be deep or shallow, depending on whether terms in the language construct synlactic or semantic representations. The deep and shallow styles are closely related, and intimately connected to folds:

1. Introduction

General-purpose programming languages (GPLs) are great for generality. Bu: this very generality can count against them: it may take a lot of programming to establish a suitable context for a particular domain; and the programmer may end up teing speilt for choice with the options available to her-especially if she is a domain specialist rather than primarily a software engineer. This tension motivates many years of work on techniques to support the development of domain specific languages (DSLs) such as VHEL, SQL and PostScript: laaguages specialized for a particular domain, incorporating the contextual assumptions of that domain and guidang the programmer specifically towards programs suitable for that

There are two main approaches to DSLs. Standalone DSLs provide their own custom syntax and semantics, and standard compilation techniques are used to translate or interpret programs written in the DSL for execution. Stardalone DSLs can be designed for maximal convenience to their intended users. But the exercise can be a significant uncertaking for the insplementer, involving an entirely separate ecosystem-compilet, editor, debugger, and 10 onand typically also much relavention of standard language features

such as local definitions, conditionals, and iteration. The alternative approach is to embed the DSL within a host GPL, essentially as a collection of definitions written in the last language. All the existing facilities and infrastructure of the host environment can be appropriated for the DSL, and familiarity with the syntactic conventions and tools of the host language can be carried over to the DSL. Whereas the standald one approach is the

Permission to make digital or hard copies of all or part of this work for personal or dassroom use is grantee without fee provided that copies are not made or datributed datatroom use is grantee without for provided that cipies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights the components of data work owned by extern that ACM results be becomed, Abstracting with credit is permitted. To copy otherwise, or republish, to practom servers on to reduce basic to lists, requires plots specific permissionmuldur a fee. Request permissione from permissions @ acts.org R.74º 74, September 1-6, 2014, Gothenburg, Sweden, Copyright @ 2014 ACM 978 1-45(3-2873-9/14/09...\$15.30.

top://cx.doi.org/10.1145/2023130.2028138

most common one within object-oriented zircles [11], the embedded approach is typically favoured by functional programmers [19]. It seems that core FP features such as algebraic data ypes and higherorder functions are extremely helpful in defining embedded DSLs; conversely, it has been said [24] that language-oriented tasks such as DSLs are the killer application for FP.

Amongst embedded DSLs, there are two further refinements. With a deep embedding, terms in the DSL are implemented simply to constract an abstract syntax tree (AST), which is subsequently transformed for optimization and traversed for evaluation. With a shallow embedding, terms in the DSL are implemented directly by their serrantics, bypassing the intermediate AST and its traversal. The names 'deep' and 'shallow' seem to have originated in the work of Boulton and celleagues on embedding hardware description languages in theorem provers for the purposes of verification [6]. Boulton's motivation for the names was that a deep embedding preserves the symactic representation of a terre, "whereas in a shallow embedding [the syntax] is just a surface layer that is easily blown away by rewriting" [5]. It turns out that deep and shallow embeddings are closely related, and intimately connected to folds; our purpose in this paper is to explore that connection.

2. Embedding DSLs

We start by looking a little closer at deep and shallow embeddings. Consider a very simple language of anthmetic expressions, involving integer constants and addition:

type $\underline{v}_{xpr_1} = ...$

-> Expr $add :: Expr_1 \rightarrow Expr_1 \rightarrow Expr_1$

The expression (3+4) + 5 is represented in the DSL by the term add (add (lit 3) (lit 4)) (lit 5). As a deep'y embedded DSL, the two operations lit and add are

encoded directly as constructors of an algebraic datatype: cata Expr2 :: * where

Lit :: Integer $\rightarrow Expr_2$ Add :: $Expr_2 \rightarrow Expr_3 \rightarrow Expr_2$ lin =Litn add xy = Add xy

(We have used Haskell's 'generalized algebraic datatype' notation, in order to make the types of the constructors Lit and Add explicit; but we are not using the generality of GADTs here, and the oldfashioned way would have worked 100.) Observations of terms in the DSL are defined as functions over the algebraic datatype. For example, here is how to evaluate an expression:

eval alg

(Add

data ExprF k where **OpF** :: $k \rightarrow k \rightarrow k$ VarF :: String \rightarrow k

language + denotational semantics

data Expr where Add :: Expr \rightarrow Expr \rightarrow Expr Var :: String \rightarrow Expr

$$\begin{bmatrix} x + y \end{bmatrix} = \begin{bmatrix} x \end{bmatrix} \bigoplus \begin{bmatrix} y \end{bmatrix}$$
(Var x) (Var y)) = eval alg (Var x) + eval alg (Var y)

eval :: (ExprF a \rightarrow a) \rightarrow Expr \rightarrow a eval alg (Op m n) = alg (OpF (eval alg m) (eval alg n)) eval alg (Var x) = alg (Var F x)

> alg :: ExprF a \rightarrow a alg(OpFab) = a + balg(VarFx) = varx





Handlers of Algebraic Effects

Matija Pretnar

Gordon Plotkin* and Matija Pretnar**

Laboratory for Foundations of Computer Science, School of Informatics, University of Edinburgh, Scotland

Gordon Plotkin

Abstract. We present an algebraic treatment of exception handlers and, more generally, introduce handlers for other computational effects representable by an algebraic theory. These include nondeterminism, interactive input/output, concurrency, state, time, and their combinations; in all cases the computation monad is the free-model monad of the theory. Each such handler corresponds to a model of the theory for the effects at hand. The handling construct, which applies a handler to a computation, is based on the one introduced by Benton and Kennedy, and is interpreted using the homomorphism induced by the universal property of the free model. This general construct can be used to describe previously unrelated concepts from both theory and practice.

Introduction

In seminal work, Moggi proposed a uniform representation of computational effects by monads [14,15,1]. The computations that return values from a set X are modelled by elements of TX, for a suitable monad T. Examples include exceptions, nondeterminism, interactive input/output, concurrency, state, time, continuations, and combinations thereof. Plotkin and Power later proposed to focus on algebraic effects, that is, effects that allow a representation by opera



Syntax represented by the free monad for a functor that provides a signature

Syntax

Semantics



Semantics often in terms of a fold over the free monad











= do or (do x \leftarrow p; k x) **do** $x \leftarrow or p q$ $(do x \leftarrow q; k x)$ k x **do** fail = **do** fail or k () fail ·Algebraic: Operations must respect substitution and sequencing with new syntactic nodes •Flexible: Multiple semantics can be given to a particular program



= do or (do x \leftarrow p; k x) **do** $x \leftarrow or p q$ $(do x \leftarrow q; k x)$ k x

> **do** fail = **do** fail k ()

 $\mathbf{do} \mathbf{x} \leftarrow \mathbf{or} \mathbf{p} \mathbf{q} = \mathbf{do} \mathbf{x} \leftarrow \mathbf{or} \mathbf{q} \mathbf{p}$

·Algebraic: Operations must respect substitution and sequencing

·Equational: The relationship between operations can be expressed by laws

with new syntactic nodes

fail

or

•Flexible: Multiple semantics can be given to a particular program



= do or (do x \leftarrow p; k x) **do** $x \leftarrow or p q$ $(do x \leftarrow q; k x)$ k x **do** fail = **do** fail or k () $\mathbf{do} \mathbf{x} \leftarrow \mathbf{or} \mathbf{p} \mathbf{q} = \mathbf{do} \mathbf{x} \leftarrow \mathbf{or} \mathbf{q} \mathbf{p}$ fail print get put throw fork read write ·Algebraic: Operations must respect substitution and sequencing •Equational: The relationship between with new syntactic nodes operations can be expressed by laws

·Pervasive: Algebraic effects cover a very large class of useful effects

•Flexible: Multiple semantics can be given to a particular program



= do or (do x \leftarrow p; k x) **do** $x \leftarrow or p q$ $(do x \leftarrow q; k x)$ k x **do** fail = **do** fail or k () $\mathbf{do} \mathbf{x} \leftarrow \mathbf{or} \mathbf{p} \mathbf{q} = \mathbf{do} \mathbf{x} \leftarrow \mathbf{or} \mathbf{q} \mathbf{p}$ fail print get put throw fork read write ·Algebraic: Operations must respect substitution and sequencing •Equational: The relationship between with new syntactic nodes operations can be expressed by laws

·Pervasive: Algebraic effects cover a very large class of useful effects

to a particular program



Handlers and Heuristics

In a tree of nondeterministic computations, there are many different evaluation strategies

For instance, breadth-first, depthfirst, depth-bounded, single result etc.

This paper shows how these different strategies are handlers of nondeterminism



This was used to model Prolog semantics as a DSL within Haskell

Heuristics Entwined with Handlers Combined

From Functional Specification to Logic Programming Implementation

Nicolas Wi

University of Oxford, UK nicolas.wu@cs.oc.ac.uk

Beroit Desouter Ghen University, Belgian moit.cesouter@agent.br

Bart Demoer KU Leuven, Belgiun bart.demoen@cs.kuleuven.b

Abstract

A long-standing problem in logic programming is how to clearly separate logic and control. While solutions exist, they fall short in one of two ways: some are too incusive, because they require significant changes to Proteg's underlying implementation, others are tacking a clean semantic grounding. We resolve both of these

We derive a solution that is both lightweight and principled. We de so by starting from a functional specification of Prolog based on monads, and extend this with the effect handlers approach to capture the dynamic search tree as syntax. Effect handlers then express heuristics in terms of tree transformations. Moreover, we can declaratively express many hearistics as trees the mselves that are combined with search problems using a generic entwining handler. Our solution is not restricted to a functional model: we show how to implement his technique as a library in Prelog by means of

Categories and Subject Descriptors D.1.1 (Programming Techmanesi: Functional Programming: D.1.6 [Programming Tech-

to intermixing and obscuring their logic with control heuristics that make finding a solution feasible. Several different solutions have been proposed to this problem. Some offer great flexibility but ahundor Prolog's execution model altogether, while those that remain faithful to Prolog offer rather limited expressive power. The Ton approach [31, 34] constitutes the current state of the art in the letter class of colutions: it is a light weight library based approach that is easily pertable to different Prelog systems. Hewever, it suffers from a major deficiency: 4 lacks proper semantic grounding, and so requires intimate knowledge of the implementation in order to be

We resolve this deficiency by borrowing techniques from functional programming-monads and effect handlers-to guide the design of a abrary that is both modular and based on sound principles. Indeed exploiting the synergy between the functional programming and logical programming paradigms is essential for this work. Because mastery of both fields is not at easy task, this synergy is narely exploited. Yet, the cross-pollization of ideas solves problems that





Unfortunately, the approach does not support syntax for constructs, which arise in a number of

state and non-determinism. The flexibility of ordering handlers is

programming with algebraic effects and handlers, we propose a catadel of abstract syntax with so-called scoped operations.

Scoped Effects

once (return x) = Just xonce (fail) = Nothing

once (or p q) = case once p of Nothing \rightarrow once q Just x \rightarrow once p

search (return x) = return x search (search p) = search p

Syntax and Semantics for Operations with Scopes Tom Schrijvers KU Leuven Belgium tom.schrijvers@cs.kuleuven.be

Mauro Jaskelioff CIFASIS-CONICET Universidad Nacional de Rosario Argentina jaskelioff@cifasis-conicet.gov.ar

structures; for example, Fiore et al. [10] model syntax with binders as algebras on certain presheaf categories.

There are two major applications of constructing formal models of syntax. First, they often lead to implementations, either in proof assistants to reason about programming calculi (for example, the



Structured Handling of Scoped Effects

Zhixuan Yang¹ 🖾 ©, Marco Paviotti¹©, Nicolas Wu¹©, Birthe van den Berg²©, ¹ Imperial College London, London, United Kingdom {s.yang20,m.paviotti,n.wu}@imperial.ac.uk ² KU Leuven, Leuven, Belgium

{birthe.wandenberg.tom.achrijvers}@kuleuwen.be offer a versatile framework that covers a





Theory Meets Practice



GitHub

Algebraic effects are nice, but we can't express lots of constructs like if statements and try/catch as syntax.

> Hmm, sounds like you need scoped effects ...

Wow! That works, but is it efficient?

Yes! It all fuses!

Amazing! 250x faster than our previous attempts! We've rolled this out to production!

3 github,	semante	Charles Flashers II (1 Security 12 Medite		
(+ Date	1) waar 80 11 P.A waart 9	Coltante Martin	al-	a warran und compa
	le anna - 1. 20 montes ()	14 mm	(D) 34-158 commits source	a codo estals meny lang
	Balandet Morat dus 1004005 PC-01	Hele Devergedon 1 -	Biertung	
		here on collision and and have been planded to be approxi-	2 poras	-
	B groups a sub-	later oper freezen de l'hornestene feiner entreferen eine prosener oan de service entre	1 HATTE	And
	B Family and the	Knop a backman started by Depayments	Antestant	Q. OW
	P 34	slarps	A section T	E en
	20 x40	lowing railing att star from photo-phate-pi-O		
	1 3X1	Includes decider Andread from theirs (party) addice	1 1011 200	Did VOU
	and a second	in northagy a Band DF	a reage at	Dia you
	a 30240	A strigt thermalia and a table	6/6/1812-3/2	a feature
	10 (DR)R	WINDOW STOCK OF AND AND A DECK	7 N/TA 48	a 100
	servante anovat	Burg second is accessible	C-northiest	dav?
	a warrants and	nort are and match	N NOTEN ARE	
	COLORE OF MIL	NUS IN FACILITY	7.00783.00	
		Burra scharke want votiens	Grantfinisji	It must
	B REALES CON	Regenerative events in the	Save Base	It must
	and and a code	Readmannics expendences	d and the bit	dout
	 semante ante 	inclusion the		uay:
	anatantic-jaan	No. of Concession, No. of Concession, Name	Time Entropy	
		Part & de la construction of the second s		

Andres Riancho

d you ever stop for a moment to think how many dev hours eature like the new @github "jump to definition" saves a

must be in the hundreds of hours across the planet per ay!



Iswim brings into sharp relief some of the distinctions that the author thinks are intended by such adjectives as procedural, nonprocedural, algorithmic, heuristic, imperative, declarative, functional, descriptive. Here is a suggested classification, and one new word.

denotative!

Peter Landin

Tunctional

Effect handlers?











References

- M. Davis, Engines of Logic, 2000
- M. Davis, The Universal Computer: The Road from Leibniz to Turing, P. Landin, A correspondence between ALGOL 60 and Church's Third Edition, 2018 Lambda-notation: Part I, 1965
- A. Hodges, Alan Turing: The Enigma, 2014
- D. Turner, Some History of Functional Programming Languages, 2012
- G. Leibniz, De Arte Combinatoria, 1666 \bullet
- D. Hilbert, W. Ackermann, Grundzüge der theoretischen Logik, 1928 R. Hinze, N. Wu, J. Gibbons, Unifying Structured Recursion J. Backus et al., The FORTRAN Automatic Coding System for the IBM Schemes, 2013
- 704, 1956
- F. Cajori, A History of Mathematical Notations, 1928 \bullet
- K. Gödel, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, 1931
- A. Church, A Set of Postulates for the Foundation of Logic, 1932
- A. Church, A Note on the Entscheidungsproblem, 1936 \bullet
- A. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem, 1937
- J. von Neumann, First Draft of a Report on the EDVAC, 1945 \bullet
- J. McCarthy et al., LISP I Programmer's Manual, 1960 \bullet
- J. Backus et al., Revised Report on the Algorithmic Language Algol 60, 1963

- P. Landin, The mechanical evaluation of expressions, 1964
- D. Scott, C. Strachey, Towards a Mathematical Semantics for Computer Languages, 1971
 - P. Landin, The Next 700 Programming Languages, 1966
- R. Hinze, N. Wu, J. Gibbons, Conjugate Hylomorphisms, 2015
- Z. Yang, N. Wu, Fantastic Morphisms and Where to Find Them, 2022
- G. Plotkin, M. Pretnar, Hanlders of Algebraic Effects, 2009
- T. Schrijvers, N. Wu, B. Desouter, B. Demoen, Heuristics Entwined with Handlers Combined, 2014
- N. Wu, T. Schrijvers, R. Hinze, Effect Handlers in Scope, 2015 \bullet
- M. Pirog, T. Schrijvers, N. Wu, M. Jaskelioff, Syntax and Semantics for Operations with Scopes, 2018
- N. Wu, T. Schrijvers, Fusion for Free, 2015
- P. Thomson, R. Rix, N. Wu, T. Schrijvers, Fusing Industry and Academia at GitHub, 2022

with thanks to Jeremy Gibbons and Jamie Willis for feedback on previous iterations of this talk

